STUDY PROJECT

THE UTILITY OF ADA FOR ARMY MODELING

BY

COLONEL MICHAEL L. YOCOM

10 APRIL 1990

U.S. ARMY WAR COLLEGE, CARLISLE BARRACKS, PA 17013-5050

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* The Utility of Ada for Army Modeling		5. TYPE OF REPORT & PERIOD COVERED Individual Study Project
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael L. Yocom, COL, SC		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army War College Carlisle Barracks, PA 17013-5050		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army War College Carlisle Barracks, PA 17013-5050		12. REPORT DATE 10 April 1990
		13. NUMBER OF PAGES 50
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Ada computer programming language was developed by the Department of Defense. The DOD has mandated its use as the single, common high-order language. A US Army Audit Agency audit of the Army Models Improvement Program found that Ada is not being used for modeling. The audit stated that, for Army modeling, either Ada must be used or a waiver must be obtained.

This study examines the Ada language and its use to date in order to judge its utility for modeling within the US Army. The study provides technical

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

and managerial evidence that Ada is potentially a very good computer language for modeling when used to support modern software engineering principles and object-oriented programming.

The study covers several major issues that impact on the decision to use Ada. It examines the potential for software reuse and portability within the modeling communities. Finally, it points out the necessity of a common language and approach for all modeling in order to meet challenges of the future family of models.

ABSTRACT

AUTHOR: Michael L. Yocom, COL, SC

TITLE: The Utility of Ada for Army Modeling

FORMAT: Individual Study Project

DATE: 10 April 1990 PAGES: 47

CLASSIFICATION: Unclassified

The Ada computer programming language was developed by the Department of Defense. The DoD has mandated its use as the single, common high-order language. A US Army Audit Agency audit of the Army Models Improvement Program found that Ada is not being used for modeling. The audit stated that, for Army modeling, either Ada must be used or a waiver must be obtained.

This study examines the Ada language and its use to date in order to judge its utility for modeling within the US Army. The study provides technical and managerial evidence that Ada is potentially a very good computer language for modeling when used to support modern software engineering principles and object-oriented programming.

The study covers several major issues that impact on the decision to use Ada. It examines the potential for software reuse and portability within the modeling communities. Finally, it points out the necessity of a common language and approach for all modeling in order to meet challenges of the future family of models.

TABLE OF CONTENTS

THE UTILITY OF ADA FOR ARMY MODELING

CHAPTER I

INTRODUCTION

Two opposing forces, declining defense resources and increasing cost, will cause the US military to turn more and more to modeling in the decade of the 1990's. Computer-assisted battlefield and weapon system simulation is more cost effective and less destructive to the environment than actual employment of weapons and vehicles. It is also more realistic than ever before. The analytic and training communities have been very successful in harnessing the power of the computer to simulate the real world. As a result, we can predict that the number, size and complexity of models will continue to grow.

There are limits to this growth, however, and they apply to all computer applications. Software currently defines the limits on the utility of the computer. Software development and maintenance costs now exceed hardware costs. System complexity and reliability problems are increasing. These general problems, coupled with our growing dependence on computer technology, have been broadly described as a software crisis. Chapter II provides the nature and history of this crisis.

Military simulation and wargaming have certainly not been exempt from the software crisis. One does not have to look far for examples. An article in Military Modeling comments on the Navy Resource Model, NARM:

> "The NARM grew so complex, so lacked documentation of many changes, and so suffered from turnover of its operator-analysts that in due course the operators were unable to justify its interactions. It had to be discarded."1

Unfortunately, the NARM project is not unique; most people involved in military modeling will know of other examples. Rather than dwell on past and existing problems, however, it is more important to see the future challenges of military modeling.

A vision for Army modeling in the decade of the 1990's includes a family of models at echelons of command from crew to theatre. Training models and simulators of different resolution, driving exercises at various echelons, will be linked, one feeding the other, and all running concurrently from widely dispersed locations. Linkage will ensure consistency of data bases and methodology.2 Similarly, a hierarchical family of combat analysis models representing various echelons and resolutions will be developed. Such linkage will not necessarily be limited to models developed by the Army. The Defense Science Board (DSB) Task Force on Computer Applications to Training and Wargaming requires an:

> "evolutionary movement (of models) toward interconnection and interoperation (that) will yield standard interfaces and standard techniques for translating data from a form suitable for one model to a form suitable for another."3

Wilbur Payne does not see this future as "an effort to develop a single, super model" but he believes agencies will accept externally guided constraints on programming standards, formats, etc.4 Obviously, a mandated modeling language is one such constraint.

BACKGROUND OF THE ISSUE

In response to growing software problems and requirements, DoD sponsored the development of the Ada programming language during the 1970's. In 1987, DoD established Ada as the single, common, computer programming language for Defense computer applications.5

The Army Model Improvement Program (AMIP) is the formal process through which the various Army agencies responsible for model development will be guided.6 A US Army Audit Agency (AAA) audit of the AMIP, dated 7 September 1989, found that the Army is not following the DoD requirement for use of Ada in Army analytical models and training simulations. The audit stated that the Army should use Ada or request a waiver.7 For US Army model clients, sponsors and analysts, therefore, the utility of Ada as a computer language for modeling is a current and relevant issue.

PURPOSE OF THE STUDY

The primary purpose of this study is to research, analyze and recommend an Army position on the use of the DoD standard programming language, Ada, as a suitable and cost effective modeling language. The technical and managerial merits of Ada as a programming language for modeling will be determined.

There are many rumors concerning Ada. This study will document the experience and lessons learned from those who have used Ada and have written about it. Ada issues that are relevant to Army modeling will be explored.

A further purpose of the study is to provide the background of Ada development and rationale for the DoD mandate. Since few decision makers and modelers in the Army's analytic and training communities have experience in the use of Ada, background information can be helpful to understanding the issue.

METHODS AND SOURCES

The study method employed is predominately qualitative. Essentially, the study involves a review of textbooks and periodical literature. Ideas, facts and issues as found in the literature are presented as they relate to the study topic. The study author draws inferences from the information presented.

Principal sources are government publications and defense related periodicals. Two college textbooks: Simulation, A Problem-Solving Approach by Stewart Hoover and Ronald Perry and Software Engineering with Ada by Grady Booch were very useful. Telephonic interviews with people currently involved in various aspects of this issue were used for background. The inferences and conclusions drawn from these sources are the author's, reflecting (limited) knowledge acquired during six years experience as a computer systems analyst plus an advanced degree in data processing.

The author is indebted to the operations research analysts of the Center for Strategic War gaming in the US Army War College for their instruction and valuable ideas. Any lack of understanding of modeling or the use of Ada on the author's part should not reflect on any of the sources mentioned.

SCOPE AND LIMITATIONS OF THE STUDY

This study is generally confined to that portion of the computer software domain that is applied to computer-assisted modeling for military analysis and training purposes.

While it is understood that there are many types and uses of military models, this study treats models in the generic sense. Specific references may not hold for the entire model taxonomy. This study is neither a rigorous examination of modeling technique nor a programming course in Ada. The syntax of the Ada language, which provides the capabilities claimed in the literature, is beyond the scope of this paper. Concepts, not code, are explored.

The scope of this study is limited to an evaluation of the Ada language against a standard of utility for modeling. This study does not evaluate other programming languages as modeling languages or in comparison to the Ada language.

This study is limited by industry's slight experience with Ada and the fact that major military modeling projects have scarcely used Ada at all. This study will not attempt to establish scientific proof of any conclusions; it establishes sufficient evidence to support inferences.

1 Wayne P. Hughes, editor, "Overview," Military Modeling, Second Edition, 1989, p. 34.

2 U.S. Department of the Army, Combined Arms Training Activity, Office of the Training Simulations Systems Manager, Family of Simulations (FAMSIM) Strategy, 5 May 1989, pp. 18-20.

3 Anita K. Jones, Chmn., Report of the Defense Science Task Force on Computer Applications to Training and War gaming, May 1988, p. 20.

4 Wilbur B. Payne, "Ground Combat Models," Military Modeling, Second Edition, 1989, pp. 41,42.

5 Department of Defense, Directive 3405.1, Computer Programming Language Policy, 2 April 1987, p. 2.

6 Department of the Army, Army Regulation 5-11 DRAFT: Army Model Improvement Program, 28 December 1988, p. 5.

7 U.S. Department of the Army, Office of the Deputy Chief of Staff for Operations and Plans, U.S. Army Audit Agency (USAAA) Audit of Army Model Improvement Program (AMIP), 7 September 1989, p. 4.

CHAPTER II

ADA BACKGROUND

Why Ada? Why Ada for Army modeling? The second question is best answered after explaining the first. This chapter explains why Ada was developed by providing the rationale, history, and policy for Ada.

THE SOFTWARE CRISIS

"Can We Trust Our Software? -- Computers are reliable but the programs that run them are fraught with peril. Just ask Ma Bell." So read the title of an article in a recent "Newsweek" magazine.1 The article said that the nine hour outage of AT&T's telephone network was caused by: "a new societal hazard of the '90's: the mysterious failure of a complicated computer software program." The article graphically portrayed the growing size and complexity of software in systems upon which the public depends and the damage brought about when software fails. The overextended, massive national air-traffic-control system was used as an example. Interestingly, reprogramming of the entire National Air System (NAS) is underway and it is being done exclusively in Ada.

Nature of the Problem

The introductory chapter and the AT&T example indicate the nature of the software crisis but some specific symptoms provided by Grady Booch are:2

- Responsiveness. Computer-based systems often do not meet user needs.

- Reliability. Software often fails.

- Cost. Software costs are seldom predictable and are often perceived as excessive.

- Modifiability. Software maintenance is complex, costly, and error prone.

- Timeliness. Software is often late and frequently delivered with less-than-promised capability.

- Transportability. Software from one system is seldom used in another, even when similar functions are required.

- Efficiency. Software development efforts do not make optimal use of the resources involved.

DoD Software Crisis

Actually, the software crisis was first publicized back in the 1970's. At that time it was estimated that up to 400 different languages were being used in DoD weapons systems.3 Each project office was virtually free to use any language, or create a new language, for its system. Since these systems are normally maintained by civilian contractors, life cycle costs became a major issue. Today, somewhere between 40% and 70% of DoD's ADP budget is spent on software maintenance. Besides being costly, language proliferation has diluted training efforts and limited technology transfer among projects.4 Lack of language standardization is not the only problem, however.

Software quality is also a major issue as "software bugs" are sometimes found long after the product is delivered, resulting in some dire consequences.

Another major problem is software maintenance. Responding to changing requirements has often meant building a new system. Old software could not be modified due to lack of modularity, poor documentation, or both. Remember the NARM cited in Chapter I?

Responding to the Crisis

Software developers must apply methods, languages, and tools to help manage complex software solutions. Booch mentions such software tools as: structured programming techniques, data-flow diagrams, object-oriented development, and integrated development environments.5 He says,

"The ultimate solution to the underlying problem of the software crisis - namely, our human limitations - lies in the application of modern software methods supported by a high-order language that encourages and enforces these principles in suitable development environments."6

ADA DEVELOPMENT

In January 1975, Malcolm Currie, Director of Defense Research and Engineering (DDR&E) formed a joint-service High-Order Language Working Group (HOLWG). The purpose of the HOLWG was to identify requirements for DoD high-order languages, evaluate existing languages against the requirements, and recommend the adoption of a minimal set of languages.7 Over 200 people, representing 85 DoD organizations, 26 industrial contractors, and 16 universities participated in the refinement of the early requirements. Most groups required a language that would support modern software engineering principles such as structured constructs, abstraction, and information hiding. Constructs for real-time control and exception handling were also required.8 The outcome of the requirements analysis phase was the conclusion that no high-order language existing at that time met the software engineering requirements. The experts deemed it necessary and feasible to develop a new language that would meet requirements.

DoD used an international competition for design of the new programming language and issued a Request for Proposal (RFP) in April 1977. Of an initial seventeen, four contractors continued the competition into the detailed design stage. In May 1979, the winner of the design competition was announced and the language was formally named "Ada" for Ada Lovelace (1815-1851), a mathematician who worked with Charles Babbage on his difference and analytic engines.9 Later in 1979, the HOLWG initiated a contract to develop a compiler validation facility and established an Ada Board to manage language changes. It approved the first version of an Ada reference manual in August 1980. This ıeference manual was approved as an ANSI (American National Standards Institute) standard on 17 February 1983. DoD establisned a policy mandating the use of Ada for all new DoD mission- critical applications beginning in 1984.

CURRENT POLICY

As established by DoD Directive 3405.1, DoD policy is to

> "satisfy functional requirements, enhance mission performance, and provide operational support through the use of ⁻ 1ern software concepts, advanced software technology, soft. ·:e life-cycle support tools, and standard programming languajes."10

Further, it is policy to limit languages to facilitate achievement of the goal of transition to the use of Ada for software development. While mandated for mission-critical applications, other applications must use Ada except where another approved language can be shown to be more cost-effective over the application's life cycle. All major upgrades in existing systems should be done in Ada.

The US Army policy and guidelines for implementing Ada as required by DoD are contained in HQDA LTR 25-88-5, dated 21 June 1988. The letter applies to all computer resources used to develop, modify, maintain, or support Army software.

[1] Michael Rogers and David L. Gonzalez, "Can We Trust Our Software?," Newsweek, 29 January, 1990, pp. 70-73.

[2] Grady Booch, Software Engineering With Ada, p. 8.

[3] "Making Ada Happen - An interview with Joseph Dangerfield," Defense Science & Electronics, December 1986, p. 30.

[4] Ibid., p. 15.

[5] Booch, p. 11.

[6] Ibid.

[7] Ibid., p. 16.

[8] Ibid., p. 17.

[9] Ibid., p. 21.

[10] U.S. Department of Defense, Directive Number 3405.1, p. 2.

CHAPTER III

MODEL DEVELOPMENT WITH ADA

Software engineering, object-oriented programming, and software reuse are not likely to be topics of conversation at a MORS (Military Operations Research Society) symposium dinner. Those who are concerned with modeling do not normally consider the "goodness" of a model to depend on the quality of the software any more than writers would judge the goodness of a book by its binding and printing. Brilliant ideas, poorly expressed, may not be read, understood, and used, however. So it is with military models whose conceptual design is extraordinary but whose software is so complex, poorly expressed and unstructured that it is not understandable, efficient, reliable, and modifiable.

Traditionally, the criteria for goodness of models have not explicitly included good software design and programming principles. Chapter one, "Overview," of Military Modeling describes good models with qualities like: robustness, predictive power, accuracy, transparency, realism, relevance, reproducible, convenience, flexibility, and sensible.1 Are these qualities sufficient?

In most military models of significance, the software design and coding of a model are done by contract for a government client or sponsor. When the computer model is fielded and used, many military analysts must learn the model, interpret its results, and either perform studies for decision makers or establish useful war games and exercises. Understandability of the computer code, therefore, is an important criteria of goodness in models. Modifiability is an additional quality required of computer models. Models are often modified to support changing requirements or to perform sensitivity analysis. Further, models must be efficient so that they provide a timely response to users who interact with them. Lastly, models should not have software bugs. They should perform as prescribed; that is, they should have the quality of reliability.

There are four properties which are recognized and accepted as goals for software engineering: modifiability, efficiency, reliability, and understandability.2 Software engineering applies a set of tools and principles to software development. It is a discipline that has been accepted by industry and DoD. The measures of goodness of models, therefore, should include sound software engineering as well as the traditional qualities. This chapter describes how Ada, plus modern software engineering and design, supports quality computer modeling. The government client should demand no less.

SOFTWARE ENGINEERING

Ada was designed to support modern software engineering.

As a design and implementation language it has many features and capabilities which make it particularly useful for large computer simulation projects. Dr. Matt Narotam, commenting on simulation issues, says the following:

"Customers like to believe that they are getting what they asked for. Even when the customer and the contractor have the best of intentions, it doesn't always work out that way.... Considering the complexity and technical sophistication of today's products, there are many reasons for this. One is that those plans aren't laid well enough that it is clear in either the customer's or the contractor's minds what really is required. Any approach, technique or tool that helps to solve this problem will prove useful and valuable. In the simulation industry ... software systems engineering and Ada will help solve the problem."[3]

Software Engineering Principles

Software engineering is defined as "the application of sound engineering principles to the development of systems that are modifiable, efficient, reliable, and understandable."[4] The software engineering principles are: abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability.[5]

The principles of software engineering apply to any software project but they seem natural to model design and development. Conceptually, models have always involved most of these principles. What may be new is that they can be applied to all steps in the modeling process: analysis, programming, testing, verification and validation.

Abstraction and Information Hiding

As mentioned before, models are abstractions; they extract essential properties from the real world. Abstraction reduces complexity. Abstraction helps in maintaining and understanding systems by reducing the details one needs to know at any given level. Furthermore, "we enhance the reliability of systems when, at each level of abstraction, we permit only certain operations and prevent any operations that violate our logical view of that level."[6] Information hiding makes inaccessible certain details that should not affect other parts of a system.

Modularity and Localization

Modularity is another tool for managing the complexity of large software systems. Modularity has been called "purposeful structuring."[7] Booch points out that higher level modules normally relate to our high-level abstractions. "A higher level module will specify what action is to be taken, while the lower level modules define how the action is to be carried out."[8] Localization is concerned with defining the interfaces with other modules in a very specific manner to support independence of the module. In a well-structured model, we should be able to understand any module relatively independently of other modules. By localizing we can limit the results of a modification to a small set of possible modules. Overall, modularity and localization support modifiability, reliability, and understandability of large models. Implemented in Ada, they support software reuse as will be discussed later.

Uniformity, Completeness, and Confirmability

Uniformity in coding style, across different submodels and modules, greatly assists in understandability of the model. The principle of uniformity calls for minimizing unnecessary differences, such as in control structure or calling sequences. This principle is particularly important for models that are used by analysts other than the builder. It is also important for the verification process. The principle of completeness means that all important elements of the problem are included in a module. "In a sense, abstraction and completeness help us develop modules that are necessary and sufficient."9 The principle of confirmability has to do with developing our system so that the parts, modules and submodels, may be tested and verified in isolation. Ada is a strongly typed language. This means that objects (nouns of the language) of a given type (set of properties) may take on only those values that are appropriate to the type and, in addition, the only operations that may be applied to an object are those that are defined for its type.10 With its strong typing, Ada can aid in making the modules of a model confirmable.

ADA FOR MODEL DESIGN

In the management of complex modeling projects, software engineering principles are important but they must be applied according to a disciplined design method. Booch gives three methods of software design as: top-down structured design, data-structure design, and object-oriented development.11 Discussion of each method is beyond the scope of this paper; however, it is important to note that most of Booch's textbook is devoted to the use of Ada in object-oriented design. This design method is particularly useful for modeling.

Object-oriented development

Using object-oriented techniques, each module in the system represents an object, (such as a truck), or class of objects, (such as vehicles), from the real world. Applied to modeling, we can map the abstractions of our conceptual model directly to modules of the computer model. We define the objects, their attributes, the operations that affect each object, the visibility of each object in relation to others, and the interfaces of objects.12 Ada may be used with any development method but, with abstraction and information hiding, it supports object-oriented development techniques quite well.

Examples of Ada modeling, using an object-oriented approach, are not readily found. However, the following description of the Army's prototype Command, Control, Communications and Intelligence, C3I, application for the Worldwide Military Command and Control System Information System, WWMCCS, could apply equally well to modeling.

"The application ... models the user's world in a software form ... Entities represent the objects that define the user's world. For example, in a tactical mission planning system, the entities might include aircraft, threats, weapons, targets, weather and intelligence. Each of the objects would have its own Ada package to define the object by its important attributes. For example, an aircraft might be described by its tail number, type, maximum range, maximum speed, maximum takeoff weight and ordnance capacity. The object-oriented approach improves system maintainability." 13

Program Design Language

Ada is not just an implementation (programming) language, but it is expressive enough to serve as a means of capturing design decisions. Many of its features are inherently useful for expressing both preliminary and detailed design form. In recent years, program design languages (PDL) have been used as an alternative to flow charts in documenting software design. PDLs improve communications among software designers, programmers, and managers (clients) through the use of commonly understood terms and concepts. DoD has mandated the use of PDLs as part of the DOD-Std-2167.

Ghazarian recommends an Ada-based PDL, Ada/PDL.[14] He points out that the use of an Ada/PDL can serve as a migration path leading to full use of Ada. Also, "Training required to use an Ada/PDL results in many people at all levels understanding the Ada language and the software engineering principles it supports." Ghazarian also mentions one PDL processor, the Ada-based design and documentation language (ADADL), which has a number of software tools specifically designed to document and assist design decisions. Its products include documents suitable for DoD/Std-2167 requirements.[15] These documents could be useful for verifying models and for training user analysts in the field.

Castor and Preston note that "Once a system is designed in an Ada PDL, the Ada definition of types and module specifications are directly used in coding and the code evolves through iterative enhancement of the design documents." [16] This fact is offered as one explanation for the observation that increased effort goes into the design phase and decreased effort goes into the coding phase of software projects when using Ada.[17]

SOFTWARE REUSE AND PORTABILITY

One of the principal requirements in the design of the Ada language was to control software costs by being portable and by supporting software reuse. Portability means that the software is not dependent on a specific run time support system (the computer platform and operating system). All certified Ada compilers must be validated periodically at a central facility in order to ensure that Ada software is portable. Additionally, the stability of the language is vigorously guarded and no subsets or supersets are tolerated.

A benefit of reuse is that rather than building up a complex system from scratch, it is possible to reuse already existing data structures and algorithms, composing systems from parts that are known to work well.18 Stability and portability of Ada program code mean Ada offers an excellent vehicle for expressing reusable software components. Additionally, certain features of the language support reuse. It has great expressive power, through a rich collection of verbs and flexible rules for noun construction, so that small modules or packages can be understood. Strong data typing, mentioned earlier, aids in readability but also helps ensure that software modules are protected from unexpected or inappropriate operations. Further, the Ada language supports structured programming; the elements of an Ada program are small Ada packages, each of which accomplishes a limited function. Ada packages could be reused by other Ada systems.

Having a library (or repository) of reusable software components, we can develop systems more rapidly since we have less software to write and we have greater confidence in the stability of the components themselves. We can also ensure common definitions of modeled objects and standard treatments of attrition, movement, etc. This could be a major advantage if we want methodologies to be parallel between all models.

Opportunities for reuse and portability

Ellis Horowitz reports that "a study done at the Missile Systems Division of the Raytheon Company observed that 40 to 60 percent of actual program code was repeated in more than one application."19 Booch cites a study by Caper Jones which concludes that of all the code written in 1983, probably less than 15 percent was unique, novel, and specific to individual applications.20 Undoubtedly, the same can be said of military models particularly in the pre- and post-processors of large systems. "After 40 years of practice, few models are designed anew: mostly they are adaptations or, for persistent problems, products of evolution."21

Wilbur Payne, noting the incentives for hardware compatibility states, "the benefits of easy software exchange between several strong research groups now greatly outweigh any benefit from keeping up with hardware state-of-the-art."22 With Ada, we can "have our cake and eat it too." It becomes possible to take advantage of the rapid progress in hardware capability, with validated Ada compilers for that hardware, and still benefit from software exchange. Given Ada code portability and machine independence, entire models could be mac[illegible]n on a variety of platforms without software modificatic.. Software reuse libraries, used elsewhere in industry and the DoD, may be the most efficient means to achieve easy software component exchange, as well as, standardization.

The Defense Science Board (DSB) Task Force Report on Computer Applications to Training and Wargaming noted the redundancy and overlap caused by each service and the JCS developing simulations independently.23 The report recommended creation of a shared simulation-data repository with encoded d[illegible] descriptions for such common information as weapon system capabilities, mobile platform capabilities, threats, e[illegible] Ada packages, modules, and programs could [illegible]ed in a central software reuse facility (l[illegible]rary/repository) for modeling. The report further recommended that the Chairman, Joint Chiefs of Staff should cause the service and joint models to interoperate and internet.[25] Software reuse permits that standardization which is a prerequisite for interoperation and internetting.

[1] Wayne P. Hughes Jr.,editor, "Overview," Military Modeling, 1989, pp. 1-43.

[2] Grady Booch, Software Engineering With Ada, p. 29.

[3] Matt Narotam, "Simulation: Getting On With Ada," Defense Science & Electronics, October 1987, p. 39.

[4] Booch, p. 42.

[5] Ibid., p. 31.

[6] Ibid., p. 33.

[7] Ibid.

[8] Ibid.

[9] Ibid., p. 35.

[10] Ibid., p. 103.

[11] Ibid., p. 36.

[12] Ibid., p. 48.

[13] Mark H. Sutton, "Army's C3I Software to Join Ada, Open Systems," Government Computer News, 13 November 1989, p.64.

[14] Saro B. Ghazarian, "Using Ada as a Software Design Tool," Defense Electronics, October 1987, p. 129.

[15] Ibid., p. 130.

[16] Virginia L. Castor and David Preston, "Programmers Produce More With Ada," Defense Electronics, June 1987, p. 168.

[17] Ibid.

[18] Grady Booch, "Reusable Software Components," Defense Electronics, May 1987, p. S57.

[19] Booch, "Reusable Software Components," p. S58.

[20] Ibid., p. S57.

[21] Hughes, p. 12.

[22] Payne, p. 141.

[23] Anita K. Jones, _Report of the Defense Science Board Task Force on Computer Applications to Training and Wargaming_, May 1988, p. 2.

[24] _Ibid_., p. 22.

[25] _Ibid_., p.21.

CHAPTER IV

ISSUES IN THE USE OF ADA

This chapter covers several managerial issues that will have a large impact on the decision to use Ada. "The decision to use Ada for a given program is thus fundamentally a business decision, not a technical one." says Major General Salisbury (US Army Ret).1 While the previous chapter was concerned with the technical decision, this chapter deals with the business decision to use Ada for modeling.

ADA COMPILER ISSUES

Availability of Compilers

The Ada Information Clearinghouse, Ada Joint Program Office, publishes the Ada Validated Compiler List monthly. The November list contained 208 base compilers and 72 compilers derived from base implementations.2 The list includes most all the recent model computers ranging from the largest mainframes, e.g., CRAY, to personal computers such as IBM PC's, PC compatibles, and Mac Intosh. The complete Digital Equipment Corporation family of VAX, VAXstation, and MicroVAX computers are supported. The popular SUN workstation has an Ada compiler. Ada compilers are now plentiful enough that availability should not preclude the use of Ada for new modeling applications. Additionally, the more popular computers have more that one compiler manufacturer.

Validation Process

For each listed compiler, the complete "Ada implementation" is validated, including the Ada compiler, linker and any other necessary software with its host computer. The validation ensures conformity of Ada implementations with the standard ANSI/MIL-STD-1815A (1983) Ada Programming Language. Conformance is measured by running a set, (called a suite), of test programs. The suite of test programs makes up the Ada Compiler Validation Capability (ACVC) which is changed every 18 months. All Ada compilers must be validated on each new ACVC in order to maintain certification.3 This process insures portability and language purity to a greater degree than any other programming language.

Speed, Efficiency, Performance

Validation ensures legitimate Ada code but it says nothing of the compiler speed and efficiency. Myers defines the criteria for compiler performance most clearly:

"To reach production quality a compiler has to translate source code to object code at a reasonable rate, it has to produce an amount of machine code that does not greatly exceed that obtained from assembly-language programming, and the execution speed of this code must be comparable to that of assembly-language machine code."4

Compiler performance is not a strength of Ada to date.

"It is currently difficult to ignore the fact that the present generation of Ada compilers usually produces code that requires between two and five times more storage (memory) and executes between two and five times slower than the equivalent code written in C (the other third-generation language)."[5]

This difference is partially offset, however, by the strong type characteristics of Ada which will catch typing errors that the programming language C, for example, will not. It should require fewer compiles with Ada to catch all errors. At any rate, small Ada modules are compiled and tested independently, perhaps at programmer workstations, and brought (linked) together at integration time. We can expect the performance of Ada compilers to drastically improve in the face of competition as the Ada industry grows.

TRAINING ISSUES

Trained Programmers

The subject of training sharply divides the pessimist from the optimist. Reporter Brad Bass, Government Computer News, says that the reason the government has been so slow in adopting Ada is, "scarcity of skilled Ada programmers, lack of federal funds for Ada training and complaints that Ada is too complex."[6] Programmers with Ada experience tend to reject government jobs because of noncompetitive salaries. Because Ada programmers are in short supply today, they are routinely offered 30 percent higher wages than non-Ada programmers in the commercial sector.[7] On the other hand, there are those who believe: "the adoption of Ada at the universities should accelerate, and by the early 1990s should provide larger pools of Ada-fluent programmers to both DoD and commercial environments."8

Gerhardt believes the real difficulty in learning Ada is that it is not like previous languages. He says:

"Not everyone can learn Ada. Ada is the first language intended for widespread usage emphasizing formality and abstraction. It is wrong to expect that everyone who is presently doing software can learn Ada."9

On the other hand, Col. James Thomes, who established the first Air Force Ada training program, strongly believes that Gerhardt and others are scaring off the very students who we need to be future programmers. Some Ada experts are making students believe that they need to know software engineering before they can understand Ada. Thomes says it is not so, "Software engineering is vitally needed, but so is a standard high-order language. And let's not merge the two so closely today that we scare off the millions of students and hackers who are going to be the programmers we need tomorrow."10

Government Analyst Training

First, we should consider who actually performs detail design and coding of the computer programs that comprise military models. If the model is a large project, chances are it is contracted to a commercial firm. Those firms will build Ada modeling expertise if their contracts specify it. In the commercial world, supply will equal demand.

There will be a need for training many analysts in the training and analytic communities of government. For the most part, however, the proficiency needed by analysts is the ability to read and understand Ada code. Some will participate in model design through an Ada-based Program Design Language. Given the expressive power of the language and good software engineering in model construction, there should not be a tremendous training burden for those who use and read models written in Ada. The government need not train analysts to be Ada programmers. Most analysts will develop considerable expertise with the language through exposure on the job. Considering that, at present, operator-analysts who use and interpret models are confronted with a variety of program languages, often in a single model, the benefit of standardization on one language will be well worth the effort to learn Ada.

SCHEDULE AND COST ISSUES

Project managers have a natural aversion to risk and they are, therefore, particularly concerned with using a new language when they have tight schedules and budgets. While the Army's long-term interest may favor accepting short-term risks for long-term reuse, maintainability, and life-cycle cost benefits, the project manager does not have the long view. The critical question on everyone's mind is whether Ada will live up to its long-term promise and justify assumption of short-term risks. Should the Army force the project manager or model sponsor to take on additional risk in the short-term in order to achieve benefits that will outlive the project or model development period?

Development Time

The results of various size projects in DoD and the commercial sector have shown, generally, that Ada developments take more time in the design phase but less time in the coding, testing, and integration phases of projects. Increased effort in the design phase has been necessary to insure good engineering, design software for potential reuse, and optimize modularity. The longer design phase, in which no code is produced and there is nothing to show, makes managers nervous. It requires faith to believe the project will come together at the late hour to make the deadline. Another generalization is that the larger and more complex the project, the more it favors Ada.

So what is the evidence that project managers or model sponsors can look to? Win Royce of Lockheed shares the following:

"One of the big problems with Fortran systems is that we typically run 15 months doing final integration and test with our big half a million lines of code plus systems. With Ada, however, the programmers integrate their code in four hours... In their private coding and testing they are getting rid of all the errors that normally plague integration and test."11

Against an accepted industry norm for programmer productivity of between 325 and 400 lines of code per programmer-month, studies show production rates with Ada of between 311 and 1400 lines of code per programmer-month.12 In addition, a considerable decrease in development time should be realized when software reuse becomes routine. Software reuse will have a major impact on development time and cost.

Life cycle costs

The promise of Ada for reducing software costs has been mentioned frequently throughout this paper because cost was a primary reason for developing the language. Design of software for maintainability, portability and reuse is focused on reduction of life cycle cost. Ralph Crafts, president of the Ada Software Alliance lobbying group, is said to have argued that "Congress could meet the Gramm-Rudman deficit reduction target simply by establishing a national software engineering policy with Ada as the standard language."13 Whether these cost savings will match the rhetoric has yet to be demonstrated. All indications are promising but few Ada projects have been around long enough to prove maintenance cost savings upon which life cycle savings are dependent.

Actually, it may not be fair to judge the merits of Ada on the earliest projects. The pioneers in the use of Ada are finding that the real benefits are not realized until the second or third project. In fact, the development costs for the first Ada project may be higher than with a familiar language due to start-up costs like training, support tools, etc.14

THE WAIVER ISSUE

The growth of the Ada industry, (producers of compilers and programmer support tools, training organizations, consulting firms, and the like), has been slower than many expected or would have liked. In a vicious circle, waivers of the DoD mandate have been awarded generously, due in large part to the non-availability of the compilers, tools, training, etc. Granting of these waivers reduced the customer base upon which the emerging Ada industry depended in order to amortize its R&D investment.15 Due to lack of investment, Ada support tools have been slow in development. Parris and Olsen comment, "Government policies were strongly worded, but compromised regularly, at the project level."16 Fortunately, it appears that a wide variety of Ada tools and services are available now. The "4th Annual Directory of Ada Tools and Services" shows that 94 companies are in the Ada business.17 The Ada industry should continue to flourish since recent indicators are that there will be far fewer waivers in the future.

Frequently, programmers, themselves, argue against an Ada mandate. "Software languages are tools and you want to use the appropriate tool," said a non-Ada programmer working on a Navy contract. "You wouldn't tell carpenters they have to use a screwdriver on every job."18 A computer scientist at the Census Bureau says that programmers will continue to use other languages unless managers push them into Ada. Further, he continues, "Nobody has said, 'Yes, I want to use Ada and train my people to use it,' hence, people have decided to use what they know best."19

Probably, the strongest argument for vigorous enforcement of the Ada mandate is standardization. Thomes provides examples of non-DoD program managers that have chosen Ada, not because Ada is so great for the particular application, but rather that it is a standard. As a standard, "it will allow software engineers to deal with millions of lines of code written by many people, on many different kinds of machines."20 From a corporate view, it may be argued that doing things the same way is more important than doing them the best way.

Another argument is that DoD has spent billions of dollars to develop Ada and make it work and DoD has an organization in place to control the language, insuring its portability, etc. DoD should orotect its investment.21 The waiver authority should beware of the low bid rationale for granting a waiver. Salisbury provides the following caution:

"The reality is that a competitively awarded contract calling for implementation of a vendor-unique and proprietary solution will generally lead to a substantial series of follow-on, sole-source procurements for system enhancements and upgrades throughout the (usually extended) system life cycle. These may, in fact, be far greater in total value than the original procurement."22

A final point to consider in granting an Ada waiver to a project manager is to realize that Ada's savings and other benefits are long-term. Unless program managers have stabilized assignments, their natural short-term risk aversion tendencies will overcome their concern for life cycle costs, reuse, and maintainability.

1 Alan B. Salisbury, "Ada and MIS," Defense Science, May 1988, p. 66.

2 Ada Information Clearinghouse, Validated Ada Compiler List, 1 November 1989, pp. 1-51.

3 Ibid, p. 57-59.

4 Ware Myers, "Ada: First users - pleased; prospective users - still hesitant," Computer, March 1987, p. 70.

5 Mark Christensen, "Is C Better Than Ada?" Journal of Electronic Defense, January 1990, p. 44.

6 Brad Bass, "Complexity Keeps Ada From Reaching Its Potential," Government Computer News, 13 November 1989, p. 67.

7 James T. Thomes, "Ada as a National Standard Programming Language," Defense Science, October 1989, p. 83.

8 Scott W. Parris and Eric W. Olsen, "The Economics of Ada," Defense Science, February 1988, p. 43.

9 Mark S. Gerhardt, "Don't Blame Ada," Defense Science & Engineering, August 1987, p. 54.

10 Thomes, p. 84.

11 Myers, p. 70.

12 Caster and Preston, p. 165.

13 Bass, p. 67.

14 Eileen Quann, "Great Expectations: Evaluating Results From First Ada Projects," Defense Science, August 1989, p. 42.

15 Parris and Olsen, p. 42.

16 Ibid., p. 41.

17 "4th Annual Directory of Ada Tools & Services," Defense Science, August 1989, pp. 44-45.

18 Bass, p. 69.

19 Ibid.

20 Thomes, p. 82.

[21] _Ibid_., p. 83.

[22] Salisbury, p. 74.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The primary purpose of this study is to research, analyze and recommend an Army position on the use of Ada as a modeling language. Research of the available literature, primarily defense periodicals, reflects the views of a broad spectrum of Ada users and other experts. Analysis of these views, coupled with textbook study, has led to the following conclusions and recommendations.

CONCLUSIONS

The bottom line conclusion to be drawn from the evidence available is that Ada is potentially a very good modeling language when used with modern software engineering principles. From a technical standpoint, Chapter III provided a description of the use of modern software engineering and object-oriented programming with Ada and related this programming to the development of military models. The research revealed no evidence that cast doubt on the suitability of Ada, with its software features and design characteristics, as a programming language for Army modeling. On the contrary, one may use the logic that follows: 1) Ada was designed to support modern software engineering and object-oriented programming, 2) Ada accomplishes these design aims in a superior fashion, 3) Object-oriented programming is commonly used for modeling, 4) Software engineering techniques are, to a degree, inherent goals of good models, 5) It follows that Ada is potentially a superior modeling language.

A further purpose of this study was to document Ada experience and lessons learned in order to identify and draw conclusions regarding what can be termed business issues. The first of these business, or managerial, issues concerns Ada compilers.

The study provides evidence to conclude that a sufficient diversity of computers have Ada compilers that this would not be a handicap in the use of Ada for modeling. Compiler performance is a concern; however, there is no evidence to determine that compiler speed would rule out the use of Ada.

Concerning training, some amount of Ada training will be required for those in the training and analytic community who have firsthand contact with models programmed in Ada. The study concludes that the benefits of a standard modeling language will offset the time and expense of one-time Ada training.

The study also concludes that Ada provides a software reuse capability which could benefit the modeling community if standard modeling procedures and data descriptions were developed and exchanged. Further, the portability of models across computer platforms by using Ada would permit a greater use and proliferation of models.

There is insufficient evidence to draw a conclusion regarding the long-term maintenance cost savings promised by Ada. Maintenance cost savings cannot be assessed until major Ada products have matured in the field. Although the potential of software reuse and long-term maintenance savings has yet to be fully realized, most authors remain of the opinion that they will occur.

Concerning the development time and scheduling implications of using Ada, the evidence shows that design will normally take longer than with other software languages. However, coding, integration and testing time will be reduced. While the internal milestones for an Ada project may change, total development time should not be longer using Ada than other languages.

Lastly, the plans for modeling in the future provide ample evidence to conclude that a degree of control and standardization will be necessary in order to avoid old problems and achieve modeling goals.

RECOMMENDATIONS

Having concluded that Ada has utility for Army modeling, from both a technical and a managerial standpoint, this study recommends its mandatory use in all major Army models. All contracts and study proposals involving software should specify Ada and software engineering standards.

Waivers should be granted rarely and only for overriding and convincing technical issues or for small time-sensitive projects that are done in-house by an analysis agency. Additionally, for those cases that warrant a waiver, it is recommended that the Army consider promoting one of the existing simulation languages, such as SIMSCRIPT II.5, as an approved high-order language. The evidence to support such a recommendation is outside the scope of this study but is valid none-the-less.

The Army Models Committee should consider the potential for software reuse in the modeling community. Use of prebuilt software, even within the Army analysis community, will require a central office of responsibility for such a reuse library and a commitment by all agencies and users to participate. That office should also have configuration management over reusable components. Software exchange could extend beyond the modeling communities to ensure consistency and efficiency in representing organizations and equipment across the Army and all services. For example, the same user entities described earlier for the Army C3I prototype could be used in all simulations to ensure commonality. The benefits to be gained: efficiency, reliability, standardization, and development time, would seem to justify the costs associated with reusable software libraries.

Concluding the adequacy of Ada for modeling and the benefit of standardization is not to say that the language should never change or improve. Ada is a third generation programming language. Fourth generation languages are entering the marketplace. In time, Ada should be upgraded to incorporate new software technologies. The Army Models Committee should be active in Ada developments to insure the language progresses to support future modeling requirements.

BIBLIOGRAPHY

1. "4th Annual Directory of Ada Tools & Services." Defense Science, August 1989, pp. 44-45.

2. Ada Joint Program Office Information Clearinghouse, Ada Validated Compiler List. The Pentagon: Nov 1989.

3. Bass, Brad. "Complexity Keeps Ada From Reaching Its Potential." Government Computer News, 13 November 1989, pp. 67-69.

4. Belanger, Ron., et al. ModSim: a Language for Object-Oriented Simulation Tutorial, CACI Products, La Jolla, CA, 30 October, 1989.

5. Booch, Grady. Software Engineering With Ada. Menlo Park: Benjamin/Cummings Publishing Company, 1987.

6. Booch, Grady. "Reusable Software Components." Defense Electronics, Vol. 19, May 1987, pp. S53-S59.

7. Castor, Virginia L. and Preston, David. "Programmers Produce More With Ada." Defense Electronics, Vol. 19, June 1987, pp. 165-171.

8. Christensen, Mark. "Is C Better Than Ada?" Journal of Electronic Defense, January 1990, pp. 44-46.

9. Dortenzo, Megan. "Ada PC Environments Offer A Change From Mainframes." Government Computer News, November 1989, pp. 63-68.

10. Firesmith, Donald G. "Should the DoD Mandate a Standard Software Development Process?." Defense Science & Electronics, Vol. 6, Part 1: April 1987, pp. 60-64, Part 2: July 1987, pp. 56-59.

11. Gerhardt, Mark S. "Don't Blame Ada." Defense Science & Electronics, Vol. 6, August 1987, pp. 53-54.

12. Ghazarian, Saro B. "Using Ada as a Software Design Tool." Defense Electronics, Vol. 19, October 1987, pp. 129-132.

13. Gordon, Geoffrey. System Simulation. Englewood Cliffs: Prentice-Hall, 1969.

14. Hayes, Richard E., and Horton, Susan M. "War Gaming, Modeling and Simulation for C2 Training." Signal, Vol. 43, No. 11, July 1989, pp. 31-35.

15. Hoover, Stewart V., and Perry, Ronald F. Simulation A Problem-Solving Approach. New York: Addison-Wesley, 1989.

16. Hughes, Wayne P., ed. Military Modeling. Military Operations Research Society, 1989. Pp. 1-43: "Overview," by Hughes and Pp. 129-144: "Ground Battle Models," by Wilbur B. Payne.

17. Joint Staff (J-8). Catalog of Wargaming and Military Simulation Models. 11th Ed. Washington, 1989.

18. Ledgard, Henry F., Ada, An Introduction. New York: Springer-Verlag, 1981.

19. "Making Ada Happen - An interview With Joseph Dangerfield." Defense Science & Electronics, Vol. 5, December 1986, pp. 30-36.

20. Miller, Howard W. "Quality Software: The Future Of Information Technology." Journal of Systems Management, December 1989, pp 8-14.

21. Myers, Ware. "Ada: First users - pleased; prospective users - still hesitant." Computer, March 1987, pp. 68-73.

22. Narotam, Matt. "Managing Ada Applications." Defense Science & Electronics, Vol. 6, May 1987, pp. 19-20.

23. Narotam, Matt. "Simulation - Getting On With Ada." Defense Science & Electronics, October 1987, pp 39-41.

24. Parris, Scott W. and Olsen Eric. "The Economics of Ada." Defense Science, Vol. 7, February 1988, pp. 41-43.

25. Passafiume, John F., Ada Education for Technical Managers. Cameron Station: Defense Technical Information Center, Defense Logistics Agency, 1981.

26. Poza, Hugo B., and Cupak, John J., "Ada: The Better Language for Embedded Applications." Journal of Electronic Defense, January 1990, pp. 47,49.

27. Quann, Eileen, "Great Expections: Evaluating Results From First Ada Projects." Defense Science, August 1989, p. 42.

28. Salisbury, Alan B. "Ada and MIS." Defense Science, Vol. 7, May 1988, pp. 66-74.

29. Shields, Jim. "Halo Delivers Colorful Graphics to Ada Compilers." Government Computer News, 13 November 1989, p. 66-67.

30. Sutton, Mark H. "Army's C3I Software to Join Ada, Open Systems." Government Computer News, 13 November 1989, p. 64.

31. Thomes, James T. "Ada as a National Standard Programming Language." Defense Science, October 1989, pp. 82-84.

32. U.S. Department of the Army, Army Regulation 5-11 DRAFT: Army Model Improvement Program. Washington: 28 December 1988.

33. U.S. Department of the Army, HQDA LTR 25-88-5, Subject: Army Implementation of the Ada Programming Language, 21 June 1988.

34. U.S. Department of Defense, Directive Number 3405.1, Subject: Computer Programming Language Policy, 2 April, 1987.

35. U.S. General Accounting Office, Status, Costs, and Issues Associated With Defense's Implementation of Ada, GAO/IMTEC-89-9, March 1989.

36. Woodward, Herbert P. "Ada - A Better Mousetrap?" Defense Science, November 1989, pp. 56,59.